Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Visakh Nair

Aligning Machine Learning for the Lambda Architecture

Master's Thesis
Espoo, September 24, 2015

Supervisor: Assoc. Prof. Keijo Heljanko, Aalto University
Advisor: Olli Luukkonen, D.Sc. (Tech.), Tieto Finland Oy

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Visakh Nair

Title:
Aligning Machine Learning for the Lambda Architecture

Date: September 24, 2015 **Pages:** 61

Major: Machine Learning and Data Mining **Code:** T-110

Supervisor: Assoc. Prof. Keijo Heljanko

Advisor: Olli Luukkonen, D.Sc. (Tech.), Tieto Finland Oy

We live in the era of Big Data. Web logs, internet media, social networks and sensor devices are generating petabytes of data every day. Traditional data storage and analysis methodologies have become insufficient to handle the rapidly increasing amount of data. The development of complex machine learning techniques has led to the proliferation of advanced analytics solutions. This has led to a paradigm shift in the way we store, process and analyze data.

The avalanche of data has led to the development of numerous platforms and solutions satisfying various business analytics needs. It becomes imperative for the business practitioners and consultants to choose the right solution which can provide the best performance and maximize the utilization of the data available.

In this thesis, we develop and implement a Big Data architectural framework called the *Lambda Architecture*. It consists of three major components, namely batch data processing, realtime data processing and a reporting layer. We develop and implement analytics use cases using machine learning techniques for each of these layers. The objective is to build a system in which the data storage and processing platforms and the analytics frameworks can be integrated seamlessly.

Keywords: machine learning, Big Data, data mining, Lambda Architecture, Internet of Things

Language: English

Acknowledgements

I would like to express my sincere and heartfelt gratitude to my thesis supervisor Assoc. Prof. Keijo Heljanko for his continuous support for my research at Aalto University. My sincere thanks goes to my thesis advisor Olli Luukkonen for his help and support.

A big thanks to Tieto Finland Oy for providing me with exciting opportunities in Big Data and machine learning research.

I thank my fellow researchers Lulit Gebremichael and Hussnain Ahmed for the thoughtful discussions we had. I would like to thank all my teachers and colleagues for their guidance and support.

Last but not the least, I would also like to thank my parents, sister and my fiancee for supporting me through this endeavour.

Visakh Nair
September 24, 2015
Espoo, Finland

Contents

Chapter 1

Introduction

The way we store, process and analyze data has undergone a huge paradigm shift in the last few years. The drastic reduction in the price of computer storage has led to huge amounts of data being retained for further analysis. This has led to the development of innovative and cutting edge technologies for storing, retrieving and analyzing data. The growing awareness of machine learning techniques has led to the development of a number of advanced analytics solutions.

As the data explosion continues, especially with the increasing usage of sensor devices, the need arises to develop an architectural framework which can cater to various analytics needs. The current Big Data software ecosystem presents a wide array of solutions and frameworks. Thus, it becomes imperative to have a good understanding of the software solutions to develop a stable umbrella framework which can cater to the data processing requirements and support advanced analytics needs.

This thesis intends to develop such a framework known as the *Lambda Architecture*. The core objectives of this thesis are the following.

- Introduce the various Big Data processing technologies and machine learning solutions.
- Discuss the machine learning analytics use cases and testing methods.
- Develop the Lambda Architecture framework and implement analytics use cases.

In addition to building a working protoype, we present multiple analytics use cases. These use cases, along with the architecture developed, helps to answer different types of business questions. For example, forecasting is a widely used technique in almost every industry. A retailer might be interested

in the sales or revenue forecast, while a home consumer might be interested in the weather forecast. Segmentation, which is inherently grouping similar items together, is another prevalent use case. In this thesis, we explore the various machine learning analytics use cases, introduce the techniques used to solve them and present how it is implemented in a Big Data environment.

Before we delve into the details, a short introduction to the various concepts explored in this thesis, namely Big Data, machine learning, Lambda Architecture and Internet of Things is provided.

1.1 Big Data

The word data is derived from the Latin word *datum*, which means *to give*. Data can be defined as a representation of a fact or idea. The origins of databases can be traced back to way before the digital age, when need arose to collect, index and retrieve data. The major turning point in databases happened in 1970, when E.F. Codd proposed the relational database model in his seminal work, *A Relational Model of Data for Large Shared Data Banks* [19]. Codd's rules, as they are known today, form the basis of relational database design.

Prevalence of cheap computational power, along with massive storage capabilities has brought a paradigm shift in the field of databases. Gartner defines Big Data as a *high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making*[1]. The advent of Big Data technologies has brought a radical change in the fields of data storage, retrieval and processing.

The need for new and improved storage and retrieval mechanisms has led to the development and adoption of NoSQL databases such as HBase, MongoDB, Riak, Cassandra and Redis and Big Data platforms such as Hadoop and Spark. Berg et al [5] provides a brief summary of the history of databases. Along with machine learning, NoSQL and Big Data platforms form the core of this thesis and are discussed in further detail in Chapter 2.

1.2 Machine Learning

Encyclopaedia Britannica describes machine learning as the discipline which concerns the implementation of software that learns autonomously[2]. In ma-

[1] http://www.gartner.com/it-glossary/big-data

[2] http://global.britannica.com/technology/machine-learning

chine learning, algorithms are developed and applied to data to build models which are used for making predictions. Data mining and pattern recoginition are two of the various facets of machine learning. Some of the common uses of machine learning in the real world are spam filters, speech recognition, computer vision, search engines and so on.

Machine learning can be broadly classified into two categories - supervised and unsupervised learning. As the name suggests, in supervised learning, labeled data is used for training the models. The data contains the attributes as well as the output class values. The algorithm infers the relation between the data vectors and their classes and generates a model which is then used for classifying new data. Classification and regression are the two major fields of supervised learning. Decision Trees, Nearest Neighbors, Random Forests, Neural Networks and Bayesian classifiers are some of the methods used in supervised learning [3, Chapter 2].

In unsupervised learning, there are no class labels present in the training data. Instead, the models are build on the data. In clustering, which is one of the most common methods in unsupervised learning, the objective is to group the objects in such a way that the objects which are similar belongs to the same cluster. It could therefore be said that it is a data driven learning mechanism [3, Chapter 7].

Since the objective of this thesis is to provide an understanding of how various machine learning methods can be utilized in Big Data platforms, it is imperative that we discuss machine learning in much detail. Therefore, we provide a very detailed and comprehensive study in Chapter 3.

1.3 Lambda Architecture

Nathan Marz[3] proposed an alternative approach to handling large scale data and coined the term *Lambda Architecture*. A system designed under the guidelines of Lambda Architecture serves as a robust and fault tolerant system which is capable of handling a wide range of workloads. The system is built in layers, with each layer performing specific functionalities and builds upon the previous one. Figure 1.1 shows the three layers or components in Lambda Architecture model. Each layer has its own defined functionalities and also builds upon the layer beneath it. The components in the Lambda Architecture model are discussed in detail in Chapters 2 and 4.

[3]https://twitter.com/nathanmarz

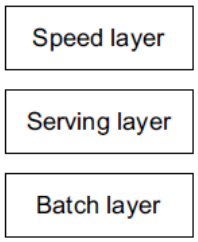

Figure 1.1: Layers of Lambda Architecture

1.4 Internet of Things

International Telecommunication Union defines Internet of Things (IoT) as *a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies*[4]. To put it simply, IoT is a network of connected devices which can communicate with each other or with a central server and transfer data and commands using industry standard protocols. These devices have been widely gaining momentum, and it has been estimated that by 2020, there will be more than 25 billion connected devices[5]. Smart homes, smart cars, agriculture, healthcare and manufacturing are some of the use cases of IoT.

The use of such devices in the manufacturing industry is commonly referred to as Industrial Internet. Predictive maintenance of manufacturing plants, logistics management etc are some examples of industrial internet in action.

In this thesis, we utilize the data from sensor devices. The device details, data model and collection methods are explored in Chapter 2. Our intention is to develop a data storage, retreival and modeling platform for this data, and we believe this will serve as a guideline for developing such a system for handling sensor data.

1.5 Structure of the Thesis

This thesis is structured as follows. In the first chapter, the goals of this thesis and the core ideas of Big Data, machine learning, Lambda Architecture

[4]http://handle.itu.int/11.1002/1000/11559
[5]http://www.gartner.com/newsroom/id/2905717

and Internet of Things are introduced. Chapter 2 introduces the three layers of the Lambda Architecture. The different software components needed to build these layers are described. In Chapter 3, the concepts of machine learning are introduced. The defining characteristics of a time series are explained. The various machine learning algorithms and statistical hypothesis testing methods used in this thesis are analyzed in detail. Chapter 4 tackles the design and implementation of a Lambda Architecture system. The implementation details of various analytical and statistical testing techniques are provided. Chapter 5 presents the details of the experiments conducted and also provides the results from each layer of the Lambda Architecture. Finally, in Chapter 6, we conclude the design and experiments conducted and mention the takeway messages from this thesis. We also look at the future of the Lambda Architecture and Big Data machine learning.

Chapter 2

Big Data

Though the term *Big Data* was initially used for huge datasets which could not be processed with traditional database technologies, nowadays it has grown to encompass a wide range of technologies, platforms and solutions. The core of Big Data solutions consists of distributed file systems such as Hadoop Distributed File System (HDFS) and parallel computing platforms such as Apache Hadoop and Apache Spark. Database solutions such as Apache Hive (relational database model, analytical processing) and Apache HBase (non-relational database model, transaction processing) serve in the framework layer. Analytics and machine learning tools such as Apache Mahout and MLlib help to bring value to the system by providing cutting edge analytical capabilities.

Since the objective of this thesis is to design and implement a Big Data machine learning system which follows the principles of the Lambda Architecture, we assume that the readers have a fundamental idea of the various terms and technologies which will be mentioned. For a concise survey, we refer to [16], and to [17] for a very detailed analysis.

2.1 Lambda Architecture

The Lambda Architecture espouses the idea that a single tool or application cannot serve as a solution for all Big Data processing problems. Instead, various tools can be utilized to build a layered solution framework, in which each layer will serve to satisfy specific needs. In this section, we will provide a bird's eye view of the Lambda Architecture. We warmly refer to [37] for a very in-depth analysis of the Lambda Architecture and its implementation strategies.

The Lambda Architecture strives to satisfy the following three design

principles.

- Fault-tolerance
- Data immutability
- Recomputation

Big Data platforms are designed to be human as well as machine fault tolerant. Data immutability means that the system will store the data but will never update data once it is written. Therefore, from the four functions in CRUD[1], we perform Create, Read and Delete, but no updates. This in turn leads to the third point, which means that it will be possible to recompute the results at any time.

The CAP theorem is a driving factor for choosing the right shared data platform for each layer in the Lambda Architecture. CAP theorem postulates that it is impossible for a distributed computer system to provide all of the following properties [8] [27]:

- Consistency (C): Each node provides the correct response to each request.
- Availability (A): Each request eventually receives a response.
- Partition tolerance (P): Properties of the system are maintained even in the case of network failures.

For consistency, the reads are guaranteed to include all previous writes and for availability, every query returns an answer. Since it is impossible to maintain all three properties in a single system, the viable designs for a distributed data platform are AP and CP (CA is not a coherent option). When choosing the platforms for each layer, we strive to maintain the CAP properties which are relevant for the purpose of that particular layer.

In the introductory chapter, the different layers of the Lambda Architecture were briefly mentioned. Now, we will discuss each of those layers in detail. We will also cite and describe the Big Data components which satisfies the functionalities of the respective layer.

[1]Create, Read, Update, Delete - Four basic functions of persistent storage

2.2 Batch Layer

The batch layer forms the core of the Lambda Architecture. Ideally, batch layer should be of high latency. This enables to perform deeper analytics and heavier computations on the data, as well as allowing full consistency and fault tolerance.

$$batch\ view = function(all\ data) \tag{2.1}$$

The Formula 2.1 defines the batch layer. It denotes that the batch layer stores the master copy of the data and computes the batch views on it. Section 1.7.1 of [37] cites the following two characteristics for the batch layer - store an immutable and constantly growing dataset, and compute functions on this dataset. From the perspective of CAP theorem, the best scenario for batch layer is consistency and partition tolerance (CP).

For readers familiar with data processing, it should be obvious that the batch layer is modeled on the traditional approach of batch data processing. The major difference is with the data being immutable, traditional batch processing applications will not suffice. Thus, the batch layer of the Lambda Architecture serves as a classical use case for the Hadoop ecosystem, which will be now discussed in further detail.

2.2.1 MapReduce

MapReduce is a programming paradigm that allows to massively scale the data processing using hundreds of servers. A MapReduce program consists of two components, a *Map* function which processes a key-value pair and generates a set of intermediate key-value pairs, and a *Reduce* function that merges the intermediate values associated with the same key [20]. The functionality of a MapReduce computation is explained below [42].

1. Each of the Map task is given one or more chunks of the file. The Map function generates the key-value pairs from this data based on the functionality.

2. The key-value pairs are collected together and sorted by key which are further divided amongst the Reduce functions.

3. For each key, the Reduce function combines the values together.

Figure 2.1 represents the above computation flow. Initially proposed by Google in 2004, MapReduce has been widely adopted by the open source community, especially the Apache Hadoop project[2].

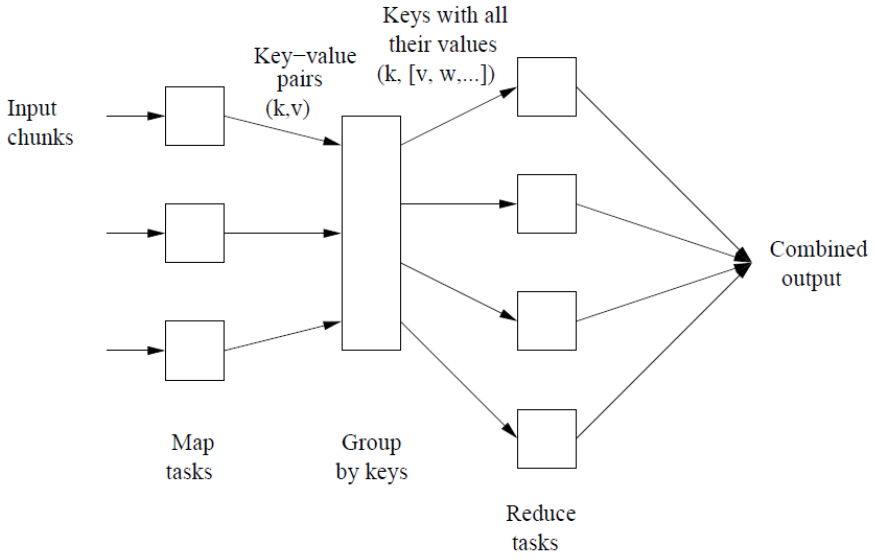

Figure 2.1: MapReduce computation flow (from [42])

2.2.2 Apache Hadoop

Apache Hadoop is an open-source project, developed and maintained by Apache Software Foundation[3], a non-profit, community-led software initiative. Hadoop framework consists of multiple components which enables distributed storage and processing of massive datasets. The two major components are:

- Hadoop Distributed File System (HDFS)
- Hadoop MapReduce

HDFS is a distributed and scalable file system. The files are divided into smaller chunks, and replicated amongst the different nodes. HDFS closely follows the Google File System, which could be seen as the precursor to HDFS [26]. Hadoop MapReduce is the open-source implementation of Google MapReduce, which has been explained in the previous section.

[2]https://hadoop.apache.org
[3]https://www.apache.org

2.2.3 Apache Hive

Apache Hive is an open-source data warehouse platform built on top of Apache Hadoop. Using Hive, analysis on massive datasets in HDFS format can be performed using SQL-like query language, known as HiveQL. Hive also enables to view the data in a more convenient relational table format. HiveQL queries are compiled into MapReduce jobs during execution. Last but not the least, Hive supports a wide array of user defined functions (UDFs), using which customized application logic can be coded and executed on Hive. For a detailed look into Hive architecture, the reader is referred to [45].

Apache Hadoop (HDFS and MapReduce) and Apache Hive form the backbone of the batch layer which is implemented in this project. The implementation details are covered in Chapter 4.

2.2.4 Alternatives

The current Big Data software ecosystem provides a wide array of solutions for building the batch layer. Another software worth mentioning is Apache Spark, which has been touted as an alternative to Hadoop. The computational model of Spark is built upon *resilient distributed datasets*, which is a collection of datasets that are partitioned across the various nodes in the cluster [50]. Spark is able to cache data in memory, thereby providing huge performance advantage for iterative machine learning computations (provided the data fits in memory). Spark SQL integrates relational data processing in Spark and can also be used to read data from Hive [4].

2.3 Serving Layer

Though Hadoop enables large scale data processing, it is not well-suited for low-latency reads, which is an often needed performance quality for reporting [33]. Therefore, it is important to design a system which enables fast and random read access to the results computed in batch layer. Serving layer bridges this gap by providing views on the batch computed data which can be queried faster.

There are a wide array of solutions available for implementing the serving layer. Marz et.al [37] recommends ElephantDB[4]. Impala by Cloudera is another open source analytics database for Hadoop[5]. In this project, we utilize

[4]https://github.com/nathanmarz/elephantdb
[5]http://impala.io

a much more traditional, but powerful and open source MySQL database.

2.3.1 MySQL

MySQL is an open-source relational database management system[6]. First developed in 1995, MySQL has come a long way and has been optimized and stabilized to support a wide array of database needs.

We chose MySQL for the serving layer for multiple reasons. Firstly, in this project, the serving layer acts mainly as a source for the processed and analyzed data. One of the major use of this data is to create visualizations. MySQL can be connected to any commercial or open-source visualization softwares without much hassle.

Secondly, from a business implementation perspective, most of the companies will have an existing relational database system. By providing the serving layer in MySQL makes the integration of the Lambda Architecture with the existing database systems smoother and easier.

In this thesis, we use a single MySQL cluster which satisfies the CP property of CAP theorem. Therefore, the system is consistent and partition tolerant.

2.4 Speed Layer

The batch and serving layers have the ability to provide most of the desired characteristics of a Big Data system, except for low latency updates. This is due to the time taken for new data to be processed and made available in the serving layer. Speed layer is designed to plug this hole by processing the realtime data.

As mentioned in Chapter 1 of [37], the realtime views in the speed layer are updated as and when new data is received. This is significantly different from the computation performed in batch layer. Therefore, it can be stated that speed layer performs incremental computation. This is defined by Equation 2.2.

$$realtime\ view = function(realtime\ view,\ new\ data) \tag{2.2}$$

Often the improved realtime performance is obtained by reducing fault tolerance to best-effort computing. This means that the speed layer generates approximations. The best effort estimates by the speed layer are replaced

[6]https://www.mysql.com

by exact values when batch layer computations are performed on the same data.

The ideal CAP properties for speed layer are availability and partition tolerance (AP). In this thesis, we equate the aspect of speed layer with stream processing paradigm. To put it concisely, we utilize a stream processing framework to process and analyze the realtime data. This is elaborated further in the below sections.

2.4.1 Apache Spark

Apache Spark is an open-source cluster computing framework for large-scale data processing[7]. An introduction to Spark architecture and processing capabilites can be found in [51]. We make use of the stream processing capabilities of Spark, which is provided as a library called Spark Streaming[8].

Spark streaming provides a mechanism to write scalable, high-throughput and fault-tolerant stream processing jobs in Scala, Java and Python. It enables to utilize the powerful semantics provided by the Spark API. The data received by Spark Streaming is divided into smaller batches, which are then processed by Spark engine to generate the output stream. Figure 2.2 shows the flow of a Spark Streaming job[9].

Figure 2.2: Spark Streaming

2.4.2 Apache Kafka

Apache Kafka is an open-source message broker application. A message broker is a programming module which translates messages from the messaging protocol of the sender to the messaging protocol of the receiver. Kafka is a publish-subscribe messaging system which can handle huge amounts of reads and writes per second from thousands of clients.[10]

[7]http://spark.apache.org
[8]http://spark.apache.org/streaming/
[9]http://spark.apache.org/docs/latest/streaming-programming-guide.html
[10]http://kafka.apache.org

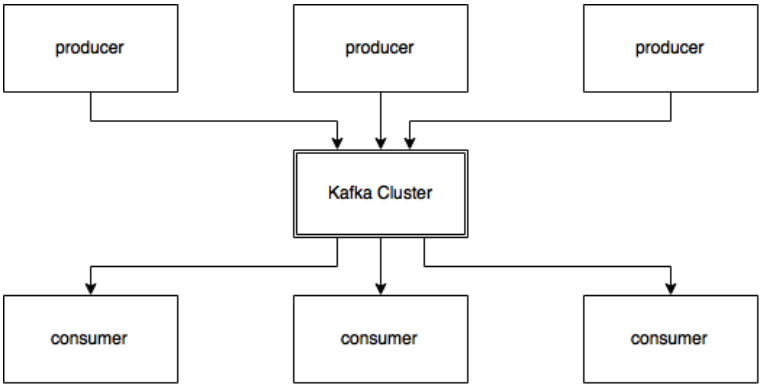

Figure 2.3: Message flow in Kafka

Figure 2.3 provides a high-level view of a Kafka system. The producers, which publishes the messages, sends them to the Kafka cluster. The Kafka cluster serves the messages to the consumers, which are the processes that subscribe to the messages. In this thesis, Kafka is utilized to collect the data from the source network and provide to Spark Streaming.

Fault tolerance in the speed layer is achieved using the message persistence feature in Kafka. Instead of deleting the messages after consumption, they are retained for a configurable period of time. If the stream processing layer misses out on any data, it can always be retrieved from the Kafka message queue.

2.4.3 Apache Cassandra

Apache Cassandra[11] is an open-source distributed database. Cassandra offers high scalability and continous availability [48]. Data retrieval in Cassandra is done using Cassandra Query Language (CQL), which is similar to SQL in syntax and usage [35]. Cassandra values availability and partition tolerance (AP)[12].

In this thesis, the realtime views in the speed layer are implemented using Cassandra.

2.4.4 Alternatives

As in the case of batch layer, an implementation of speed layer can also be developed using different software components. Apache Storm is a distributed

[11]http://cassandra.apache.org
[12]https://wiki.apache.org/cassandra/ArchitectureOverview

realtime computing system. Real-time analytics and online machine learning are two of the various use cases that can be implemented in Storm. Storm is scalable and fault-tolerant with at least once message processing [46]. Trident, which provides a higher level abstraction on Storm and exactly once processing, can be utilized to mix high throughput stream processing with low latency distributed querying [9]. Storm, together with Trident, can serve as an alternative to Spark Streaming.

Apache HBase and Riak are two alternatives to Cassandra for developing the views in the speed layer. HBase is a non-relational distributed database modeled after Google's BigTable [14] [24]. HBase is consistent and partition tolerant (CP). Riak, which is modeled after Amazon's Dynamo, is a distributed key-value datastore [12]. Riak supports high availability and is partition tolerant (AP).

2.5 Visualizations

The real value from any data analytics solution is derived by the judicious and innovative use of visualizations. In this thesis, we utilize the plotting functionalities of R graphics package [41] and Tableau Desktop 9.0. Tableau Desktop is a professional data visualization platform[13]. It has been rated as one of the top five data visualization tools by Gartner [43]. Tableau also offers native drivers to connect to a wide variety of Big Data platforms such as Spark SQL, Hadoop, Hive, MySQL and so on.

2.6 Data

Table 2.1: Sensor and lane

Sensor id	*Lane details*
4	North-South, right lane
5	North-South, middle lane
6	North-South, left lane
7	South-North, right lane
8	South-North, left lane

The data processing and analysis in this thesis is performed on time series data (see Section 3.2 for some of the defining aspects of a time series). The

[13]http://www.tableau.com/products/desktop

time series data is collected from various traffic sensors installed in a five lane road in a city in Northern Finland. Each lane has its own sensors and collect various attributes. The placement of the sensors and the lanes is summarized in Table 2.1.

The attributes of interest are listed in Table 2.2.

Table 2.2: Data Attributes

Attribute name	*Description*
Sensor ID	Distinct id for each sensor, starting from 4 till 8
Speed	Speed of vehicle in km/h
Height	Height of vehicle in centimeters
Length	Length of vehicle in meters
Snow Depth	Amount of snow on road in millimeters
Timestamp	Time when vehicle passed the sensor

The data is collected by Noptel Oy[14] and provided for usage through a web interface. A *cron* job queries the web server, collects the data and writes to a file. The data from these files are used for further analysis so that the web server is not overloaded.

[14]http://www.noptel.fi/eng2/index.html

Chapter 3

Machine Learning

In this chapter, we discuss the details of various machine learning methods implemented in this thesis and also provide an overview of how these methods can be integrated into the Lambda Architecture framework.

3.1 Introduction

As the title of this thesis suggests, the main aim of this work is to design, develop and implement a Lambda Architecture framework and perform analysis on the data using various machine learning algorithms. The previous chapter provided a discussion about the various components of the Lambda Architecture and the various solutions which were chosen in this thesis for implementing it.

The intention of this chapter is to explain the various machine learning and statistical methods that will be utilized in this thesis. We also look at the various statistical hypothesis tests conducted. The major software platforms/solutions utilized are also introduced and explained.

Machine learning is a very vast field, with potentially many use cases. In this chapter, the literature analysis is limited to the use cases which are explored and experimented in further chapters. We assume that the reader is familiar with some of the basic aspects of machine learning. We recommend [6] and [3] for exploring machine learning further.

As mentioned in Chapter 2, the data utilized in this thesis is a time series. A time series can be defined as a collection of data points recorded over a period of time such as seconds, minutes, hour, day, month and so on. In the next few sections, we will describe the various analysis, forecasting and statistical testing methods which can be used for time series data modeling. We will also describe the machine learning algorithms and techniques utilized.

For a much detailed text on time series modeling, we refer to [10].

3.2 Time Series

A time series is a set of observations x_t, recorded at time t. For example, the stock prices of a particular company can be represented as a time series. The period of time can vary based on how frequently the data is collected. For the stock price time series, it might be beneficial to collect the price every minute, but if we are constructing a time series of temperature recorded, it might be useful to collect the data every hour.

A time series can be said to be composed of the following four different components:

1. *Secular Trend* - Secular trend can be defined as the smooth long term trends in a time series, which can be increasing, decreasing or staying the same over time.

2. *Cyclical Variation* - Cyclical variation refers to the rise and fall of a time series over longer periods, predominantly more than a year.

3. *Seasonal Variation* - It refers to the pattern of changes in a time series within a year.

4. *Irregular Variation* - It can be further subdivided into episodic and residual variations. Episodic variations are unpredictable and can be identified, whereas residual variations are unpredictable and cannot be identified.

Before getting into the statistical testing and forecast modeling, we would like to present a few characteristics associated with time series data.

3.2.1 Stationarity

A time series can be defined as a stationary time series if the properties do not depend on the time at which the series is observed [31]. In other words, the joint probability distribution of the series does not change when shifted in time. The statistical properties such as mean, variance and so on will remain constant over time for a stationary time series. A stationary time series will not have any predictable patterns in the long run. Therefore, if the time series expresses seasonality or trend, then it is not stationary.

It is important to note that most of the forecasting methods assume the time series to be stationary. So, it is imperative to test for stationarity

and take appropriate measures to remove if any non-stationarity is present. Differencing is one solution and is presented next.

3.2.2 Differencing

The process of differencing involves computing the differences between consecutive observations. Differencing helps to stabilize the mean of the series and eliminates trend and seasonality. The order of differencing refers to the number of times the series has been differenced. For example, in a second order differenced time series, the data is differenced a second time in order to make it stationary.

3.3 Descriptive Statistics

Descriptive statistics refer to the various methods using which the data can be summarized in a meaningful way so as to understand the quantitative information present in the data. These are the methods that can be utilized to understand, summarize and visualize the data. In this thesis, we analyze the *moving average*, which is a measure of central tendency, and *moving standard deviation*, which is a measure of spread.

3.3.1 Moving Average

Moving average is a statistical method of creating a series of averages of subsets of the original time series. For a given time series

$$Y = \{y_1, y_2, ..., y_T\}, T \in \mathbb{Z} \tag{3.1}$$

the n moving average is computed by taking the arithmetic mean of n subsequent terms of the series. The first element of the moving average is computed by taking the arithmetic mean of the first n elements in the series. The series is shifted forward, excluding the first element of the series and including the next element, and computing the mean again. This is performed for the entire series.

For the series defined by Equation 3.1, the moving average can be defined as:

$$\mu_t(n) = \frac{1}{n} \sum_{i=0}^{n-1} y_{t-i} \tag{3.2}$$

where n denotes the rolling window size.

Moving average is used to smoothen the fluctuations in the data. The resulting series helps to identify the trends and seasonalities in the data. Though the concept behind moving averages is simple enough, they form a crucial basis for more complex time series analysis techniques such as forecasting and decomposition [52].

3.3.2 Moving Standard Deviation

Standard deviation is a measure of the spread of values in a series. It measures the amount of variation of the values in a dataset. In other words, standard deviation can be defined as how tightly the values are clustered around the mean.

Standard deviation is calculated by taking the square root of the averages of the squared deviations of values from their mean. The moving variance is computed using Equation 3.3 and moving standard deviation using Equation 3.4.

$$\sigma_t^2(n) = \frac{1}{n-1}\sum_{i=0}^{n-1}(y_{t-i} - \mu_t(n))^2 \tag{3.3}$$

$$\sigma_t(n) = \sqrt{\sigma_t^2(n)} \tag{3.4}$$

3.4 Forecasting Models

Time series forecasting involves the creation of a model to predict the future values based on the previous values. Forecasting methods itself is a field of its own. In this section, we will consider the forecasting models which were used for the prediction tasks. For an in-depth analysis of these models, the reader is referred to Chapter 6 of [11].

3.4.1 Moving Average Models

A moving average (MA) model is one of the simplest models in time series forecasting. It is a finite-impulse response filter applied to white noise. In other words, a MA model is a linear combination of white noise processes so that the variable y_t depends on the current and previous values of a white noise disturbance term. A moving average process of order q is represented by Equation 3.5.

$$y_t = \mu + u_t + \theta_1 u_{t-1} + \theta_2 u_{t-2} + ... + \theta_q u_{t-q} \tag{3.5}$$

where u_t is a white noise process with zero mean and constant variance and $\theta_1, \theta_2, ..., \theta_q$ are the model parameters.

3.4.2 Autoregressive Models

In an autoregressive (AR) model, the forecast value of the variable depends on the past value of the variable and a stochastic error term. For a variable y, the autoregressive model of order p can be written as,

$$y_t = \mu + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + u_t \tag{3.6}$$

where u_t is the white noise disturbance term and $\phi_1, \phi_2, ..., \phi_p$ are the model parameters. Equation 3.6 is a stochastic difference equation.

3.4.3 ARMA Models

Autoregressive moving average models, better known as $ARMA(p, q)$, is obtained by combining $AR(p)$ and $MA(q)$ models. An ARMA model explains a weakly stationary stochastic model using an autoregressive polynomial and a moving average polynomial. In ARMA, the current value of the variable y is linearly dependent on the previous values as well as a combination of current and previous values of the white noise error term. An $ARMA(p, q)$ model is represented by Equation 3.7.

$$y_t = \mu + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \theta_1 u_{t-1} + \theta_2 u_{t-2} + ... + \theta_q u_{t-q} + u_t \tag{3.7}$$

It can be readily noted that Equation 3.7 is a combination of $AR(p)$ and $MA(q)$ models defined by Equations 3.6 and 3.5 repectively. Therefore, the characteristics of an $ARMA$ model will be a combination of the characteristics of its AR and MA parts. To fit an ARMA model to a time series, the p and q values need to be computed. Once the values are chosen, the parameters can be calculated by a least squares regression.

3.4.4 ARIMA Models

Another widely used forecasting method is autoregressive integrated moving average model, better known as $ARIMA(p, d, q)$. It is a generalization of $ARMA(p, q)$ model in which the variable is differenced d times. ARIMA models are useful especially in cases where the time series data is non-stationary.

In such cases, differencing the data helps to reduce the non-stationarity [31]. The model is represented by the Equation 3.8:

$$y'_t = \mu + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + ... + \phi_p y'_{t-p} + \theta_1 u_{t-1} + \theta_2 u_{t-2} + ... + \theta_q u_{t-q} + u_t \quad (3.8)$$

where y' represents the differenced time series, $\theta_1, \theta_2, ..., \theta_q$ are the moving average parameters and $\phi_1, \phi_2, ..., \phi_p$ are the autoregressive parameters.

3.5 Statistical Hypothesis Testing

In a big dataset (called population in statistical testing circles), it is impractical to observe every single observation. So it is necessary to sample the population and use that data to understand the population and answer questions about it. Hypothesis testing is a statistical method which helps to draw inferences about the population from sample data.

Gravetter and Wallnau [28] defines hypothesis testing as *a statistical method which uses sample data to evaluate a hypothesis about a population.* The steps involved in a hypothesis testing exercise are as follows.

1. A hypothesis about the population is formulated. A *null* (represented as H_0) and an *alternative* (represented as H_1) hypothesis are formed. The null hypothesis is a claim of no difference and the alternative hypothesis is a claim of difference in the population [25].

2. Using the hypothesis, predict the characteristics of the sample.

3. Select a random sample from the population.

4. Compute the test statistics. There are various types of test statistics and they are discussed in the upcoming sections.

The decision to reject or not reject the null hypothesis is made based on either the *p-value* or *region of acceptance*. The *p-value* measures the strength of evidence supporting the null hypothesis. Region of acceptance is a range of values, and if the test statistic falls within it, the null hypothesis is not rejected.

A *one-tailed* test is one in which the region of rejection is only on one side of the sampling distribution, whereas a *two-tailed* test is one in which the region of rejection is on both sides of the sampling distribution.

Chi-squared statistic is a commonly used test statistic. It follows a *Chi-squared* (χ^2) distribution. A *Chi-squared* distribution with k degrees of freedom is the distribution of the sum of squares of k independent standard normal variables [2, Chapter 26].

The details of the hypothesis tests used in this thesis are discussed next.

3.5.1 Tests of Normality

Normality tests are used to determine whether the population (dataset) is normally distributed or not. Testing for normality is important since most of the statistical tests assume that the underlying data follows a normal distribution.

A normal distribution is a continuous probability distribution defined by Equation 3.9.

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.9}$$

where μ is the mean and σ is the standard deviation.

The third and fourth moment of the distribution are skewness and kurtosis respectively (first and second moments being mean and variance). These higher moments are needed to characterize the distribution. As the name suggests, skewness defines the shape of the distribution and measure the extent to which the distribution is symmetric to its mean. Kurtosis measures the fatness of the tail of the distribution. A normal distribution is not skewed and is defined to have a kurtosis of 3 [11, Chapter 2].

Normal distribution is commonly known as *bell curve* due to its bell shaped structure. But since there are other distributions which also take a bell shape, it is imperative to have a statistical test to validate the normality of the data. Jarque-Bera test is used for normality testing in this thesis.

3.5.1.1 Jarque-Bera Test

Jarque-Bera (sometimes also refered to as Bera-Jarque) test is developed based on the property that a normal distribution is characterized by the first and second moments. Jarque-Bera test defines the coefficients of excess kurtosis and skewness and tests whether they are jointly zero. The null and alternative hypothesis are as follows.
Null Hypothesis, H_0 : Normal distribution
Alternative Hypothesis, H_1 : Non-normal distribution

The coefficients of skewness and kurtosis can be defined as:

$$b_1 = \frac{E[u^3]}{(\sigma^2)^{\frac{3}{2}}} \tag{3.10}$$

$$b_2 = \frac{E[u^4]}{(\sigma^2)^2} \tag{3.11}$$

where E denotes the expectation.

Since the kurtosis for a normal distribution is 0, the excess kurtosis is $b_2 - 3$. For a sample of size T, the Jarque-Bera statistic is given by:

$$W = T\left[\frac{b_1^2}{6} + \frac{(b_2 - 3)^2}{24}\right] \tag{3.12}$$

Interested readers are referred to [32] for a derivation of this statistic.

The test statistic, W, is compared with a *Chi-squared* ($\chi^2(2)$) distribution with two degrees of freedom. The null hypothesis of normality is rejected if the test statistic exceeds the critical value. Table 3.1 shows the critical values for a *Chi-squared* distribution with two degrees of freedom.

Table 3.1: Critical values of *Chi-squared* distribution

Significance level	*Critical value*
0.10	4.61
0.05	5.99
0.01	9.21

3.5.2 Tests of Stationarity

The concept of stationarity in a time series was discussed in Section 3.2.1. This section will discuss the various statistical testing methods used to check for the presence or non-presence of stationarity in a time series. Before getting into the details of the test, we need to define *unit root*. Unit root refers to the existence of characteristic equation of a time series model on the unit circle. If the characteristic equation of a linear stochastic process has a root of 1, it is said to have a unit root.

3.5.2.1 Dickey-Fuller Test

The null and alternative hypothesis for Dickey-Fuller test are as follows.
Null Hypothesis, H_0 : Series contains a unit root
Alternative Hypothesis, H_1 : Series is stationary

The regression is defined by Equation 3.13.

$$\Delta y_t = \psi y_{t-1} + u_t \tag{3.13}$$

The test statistic for Dickey-Fuller test is defined as:

$$test\ statistic = \frac{\hat{\psi}}{SE(\hat{\psi})} \quad , \tag{3.14}$$

where SE is the standard error.

The objective of Dickey-Fuller test is to check the null hypothesis that $\psi = 1$ in

$$y_t = \psi y_{t-1} + u_t \tag{3.15}$$

The critical values for the Dickey-Fuller test statistic are derived from simulation experiments. The null hypothesis is rejected if the test statistic is more negative than the critical value [22] [11, Chapter 13]. The Dickey-Fuller critical values for sample size of 25 are shown in Table 3.2.

Table 3.2: Dickey-Fuller critical values for sample size of 25

Significance level	*Critical value*
0.10	-1.60
0.05	-1.95
0.01	-2.65

3.5.2.2 Augmented Dickey-Fuller Test

Standard Dickey-Fuller test does not account for autocorrelation in the data. The augmented Dickey-Fuller test provides an alternative model as shown in Equation 3.16.

$$\Delta y_t = \psi y_{t-1} + \sum_{i=1}^{p} \alpha_i \Delta y_{t-1} + u_t \tag{3.16}$$

The lags defined by Δy_t ensures that u_t is not autocorrelated. The critical values for the augmented Dickey-Fuller test are the same as that of Dickey-Fuller test.

3.5.3 Error and Residual Analysis

When modeling a regression, it is important to assess the appropriateness of the model. It is imperative to check how well the model explains the data. To perform this analysis, there are various statistical testing techniques available. This section covers the tests that are used in this thesis.

In regression analysis, the following assumptions play a pivotal role in understanding the adequancy of the model (u_t is the error term) [11, Chapter 5].

1. All the relevant variables are included in the model.
2. The functional form is correct.
3. The random error term has zero mean ($E(u_t) = 0$) and constant variance ($var(u_t) = \sigma^2 < \infty$).
4. The errors are not uncorrelated ($cov(u_i, u_j) = 0 \; for \; i \neq j$).

If the variance of the errors is constant, it is known as homoscedasticity. Heteroscedasticity refers to the case when the errors do not have a constant variance. Goldfeld-Quandt test (see Section 3.5.3.1) checks for heteroscedasticity in the error terms. Ljung-Box test (see Section 3.5.3.2) checks for autocorrelations in a time series. Residual plots (see Section 3.5.3.3) provide a graphical method to perform the validation of the model.

3.5.3.1 Goldfield-Quandt Test

As mentioned earlier, Goldfeld-Quandt test is a test for heteroscedasticity. In Goldfeld-Quandt test, the data sample of length T is split into two sub-samples of length T_1 and T_2. The regression model is estimated on each of these sub-samples. The residual variances are calculated by the following equations (here k is the number of parameters in the model).

$$s_1^2 = \frac{\hat{u_1}'\hat{u_1}}{T_1 - k} \tag{3.17}$$

$$s_2^2 = \frac{\hat{u_2}'\hat{u_2}}{T_2 - k} \tag{3.18}$$

The null and alternative hypothesis are as follows:
Null Hypothesis, H_0 : The variances are equal
Alternative Hypothesis, H_1 : The variances are not equal

The test statistic is computed by [11, Chapter 5]:

$$GQ = \frac{s_1^2}{s_2^2} \tag{3.19}$$

The test statistic follows an F distribution with $(T_1 - k, T_2 - k)$ degrees of freedom. Some sample values of the F distribution are shown in Table 3.3.

Table 3.3: F distribution values for significance level of 0.05

Degrees of Freedom	*Critical value*
(2,2)	19.00
(5,5)	5.05
(10,10)	2.98

3.5.3.2 Ljung-Box Test

Ljung-Box test examines the autocorrelations of the residuals. The hypothesis are as follows:
Null Hypothesis, H_0 : The model does not exhibit lack of fit
Alternative Hypothesis, H_1 : The model exhibits lack of fit

For a time series of length T, the Ljung-Box test statistic is defined as [11, Chapter 6]:

$$Q = T(T+2)\sum_{k=1}^{m}\frac{\hat{\tau}_k^2}{T-k} \tag{3.20}$$

where $\hat{\tau}_k^2$ is the estimated autocorrelation of the series at lag k and m is the number of lags being tested.

The Ljung-Box statistic follows a $\chi^2_{(m)}$ (*Chi-squared*) distribution.

3.5.3.3 Residual Plots

Residual plots are a graphical way to identify whether there are any relationships between the error terms. The current residual and the immediately previous residuals are plotted over time. Such a plot will help to identify one of the three following possible patterns.

1. Positive autocorrelation: If the residual at time t - 1 is positive, then the residual at time t is also positive.

2. Negative autocorrelation: If the residual at time t - 1 is positive, then the residual at time t is negative and vice-versa.

3. No autocorrelation: In this case, there is no pattern in the residuals and the points are randomly spread.

In addition to the graphical methods, it is also possible to check for autocorrelations using statistical testing. Durbin-Watson test is such a method and is discussed next.

3.5.3.4 Durbin-Watson Test

Durbin-Watson (DW) tests for first order autocorrelation in the residuals (relationship between error and its immediately previous value). The hypothesis are:
Null Hypothesis, H_0 : No autocorrelation present, $\rho = 0$
Alternative Hypothesis, H_1 : Autocorrelation present, $\rho \neq 0$

Durbin-Watson test statistic is computed by the Equation 3.21:

$$DW = \frac{\sum_{t=2}^{T}(\hat{\mu}_t - \hat{\mu}_{t-1})^2}{\sum_{t=2}^{T}\hat{\mu}_t^2} \tag{3.21}$$

where T is the sample size.

The DW statistic can also be expressed in terms of ρ as:

$$DW \approx 2(1 - \hat{\rho}) \tag{3.22}$$

The DW statistic can take values between 0 and 4. The two critical values are the *upper* critical value (d_U) and the *lower* critical value (d_L). The rejection and non-rejection regions are shown in Figure 3.1.

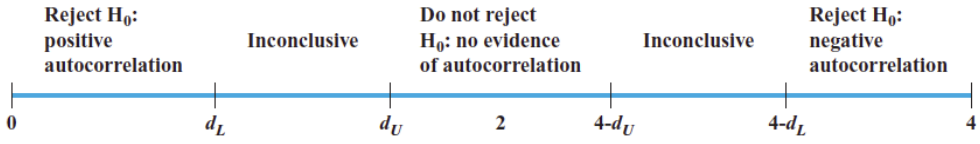

Figure 3.1: Rejection and non-rejection regions of DW test [11, Chapter 5]

The Durbin-Watson test statistic can be summarized as follows.

1. DW $= 0$: Perfect positive autocorrelation in the residuals
2. DW $= 2$: No autocorrelation in the residuals
3. DW $= 4$: Perfect negative autocorrelation in the residuals

3.6 Clustering

Clustering is an unsupervised machine learning technique in which the objective is to group similar objects together. In clustering, the aim is that similar objects are closer to each other than dissimilar objects. It is an unsupervised learning technique since there is no labeled data assigning an object to a cluster.

In this section, the clustering methods utilized in this thesis are discussed briefly. For a comprehensive study on clustering, we refer the reader to [3, Chapter 7] and [6, Chapter 9].

3.6.1 *k*-means Clustering

k-means clustering is one of the most popular clustering methods. In k-means, the primary objective is to cluster the given set of data into k clusters, where k is pre-defined or fixed *a priori*. The algorithm tries to find the best k grouping for the data so that the total distance between the group members and the group's centroid is minimized. This criteria can be written as:

$$\sum_{j=1}^{k}\sum_{i=1}^{n}|x_i^j - c_j|^2 \tag{3.23}$$

where $x_1, x_2, ..., x_n$ are the data to be clustered and $c_1, c_2, ..., c_k$ are the cluster centroids.

The various steps in k-means algorithm are as follows.

1. Define the initial centroids. A common way to begin with is to assign random values.
2. Assign each object to the cluster which has the closest centroid. This is performed by computing the distance between the object and the centroid.
3. Recompute the centroid values based on the clusters formed.
4. Repeat steps 2 and 3 iteratively till there is no cluster reassignment for the objects.

Finding the exact solution for the k-means problem for an arbitrary input is *NP*-hard [23]. Therefore, the solutions to k-means are usually heuristic algorithms which converge to a local optimum. Lloyd's algorithm is the most popular algorithmic implementation of k-means [34].

The number of clusters, k, can be set using various methods. In this thesis, the *elbow-method* using sum of squared errors (SSE) is made use of. SSE is the sum of the squared distance between each cluster member and its cluster centroid. The idea is to choose the best number of clusters so that adding another cluster does not improve the model. The SSE for different number of clusters is plotted. The cluster size for which the reduction in SSE slows dramatically is chosen as the k value [29].

3.6.2 Streaming k-means Clustering

The k-means algorithm described in Section 3.6.1 is generally used to cluster already present data. In other words, it can be used to compute the clusters in batch mode, on a previously collected dataset. But when it comes to real time data analysis, the need arises for clustering dynamically. A straightforward approach is to build the model based on the static data and use it to predict the incoming real time data. But this approach is not sufficient if the patterns in the data keeps on changing by time. In such a case, a mini-batch k-means algorithm can be utilized. In this, the k-means steps are performed for each batch of data. The cluster centers computed are used to perform k-means clustering on the new batch of data [7].

One important aspect to consider in streaming k-means is the importance of new data versus old data. It is imperative to balance the importance of new data against past data. This is handled by a parameter in the k-means, α. If α is set to 0, then only the most recent data will be used. If α is set to 1, then the whole available data is used. Equations 3.24 and 3.25 denote how the cluster centers are updated[1].

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t} \tag{3.24}$$

$$n_{t+1} = n_t + m_t \tag{3.25}$$

where c_t denotes the previous centroid of the cluster, n_t is the number of points assigned to the cluster, x_t is the new centroid of the cluster computed in the current batch, m_t is the number of points added to the cluster in the current batch and α is the decay factor.

3.6.3 Expectation-Maximization

Expectation-Maximization (commonly known as *EM*) algorithm is an iterative method to find the maximum likelihood estimate of the parameters. It

[1] http://spark.apache.org/docs/latest/mllib-clustering.html

involves two sets of random variables of which one is observable(X) and the other is hidden(Z). The *EM* algorithm tries to find the parameter vector P that maximizes the likelihood of observed values of X, $\mathcal{L}(P|X)$. The hidden variable Z is also utilized to generate a joint distribution, $\mathcal{L}(P|X, Z)$.

The Expectation step (E) creates a function for the expectation of the log-likelihood, and the Maximization step (M) computes the parameters by maximizing the expected log-likelihood computed in the E-step. If $\mathcal{L}_c$ denotes the complete likelihood, Q denotes the expectation and l denotes the index of iteration, then the E and M steps can be defined by Equations 3.26 and 3.27 respectively.

$$E - step : Q(P|P^l) = E[\mathcal{L}_c(P|X, Z)|X, P^l] \tag{3.26}$$

$$M - step : P^{l+1} = \arg\max_P Q(P|P^l) \tag{3.27}$$

For the derivation and further explanation, the reader is referred to [3, Chapter 7] and [21].

3.7 Technology

In this section, the details of the technology platforms and libraries used for machine learning in this thesis are discussed. This is intended to be a concise and quick introduction to the relevant platforms. For further details, we refer the reader to [49] and [38].

3.7.1 R

R is a programming language which is predominantly used for statistical computing. Using R, a varied array of statistical and machine learning tasks such as time series analysis, classification, clustering, graphical techniques and so on can be performed. The power of R comes with the hundreds of community developed packages.[2]

In this thesis, we utilize many R packages to perform various machine learning tasks and statistical hypothesis testing. Table 3.4 lists the package names and its version details [30] [47] [41] [36] [39] [13] [15] [18].

[2]https://www.r-project.org/about.html

Table 3.4: R packages

Package Name	*Version*
forecast	5.9
tseries	0.10-34
stats	3.2.2
cluster	2.0.1
RHive	2.0-0.2
TSA	1.01
shiny	0.11.1
EMCluster	0.2-4

3.7.2 MLlib

MLlib is an open-source distributed machine learning library which is part of Apache Spark[3]. MLlib includes a wide array of machine learning algorithms, notably classification, clustering, regression and collaborative filtering. MLlib also provides a wide variety of underlying statistics, linear algebra and optimization primitives. MLlib utilizes the scalable processing power of Spark and thereby enables high speed and large scale learning. MLlib also provides APIs that operate with Scala, Java and Python [38].

[3]http://spark.apache.org/mllib/

Chapter 4

Lambda Architecture

In Chapter 2, the three layers of the Lambda Architecture were introduced and explained. The various software components for each of these layers were also discussed. Chapter 3 provided a detailed look into the various statistical analysis and machine learning methods. This chapter will provide the implementation details for the different layers in the Lambda Architecture, as well as the details on the statistical analysis and machine learning use cases.

The architecture diagram for the Lambda Architecture model developed in this thesis is shown in Figure 4.1. It shows the different components which builds the three different layers of Lambda Architecture. The various software components which are utilized to develop the functionality of each layer are also shown.

4.1 Batch Layer

The batch layer is implemented using Apache Hadoop and Apache Hive. All the data collected from the sensors (see Section 2.6) is utilized in this. In the upcoming sections, the descriptive statistical analysis, speed forecasting and the clustering of vehicles are explained.

4.1.1 Implementation

As mentioned earlier, the core of batch layer consist of Apache Hadoop and Apache Hive. The data collected from the sensors is loaded into Hive tables, with some transformations done. A staging-target table architecture is followed. The process flow is shown in Figure 4.2.

The following analytical tasks are performed on the data loaded into the target table.

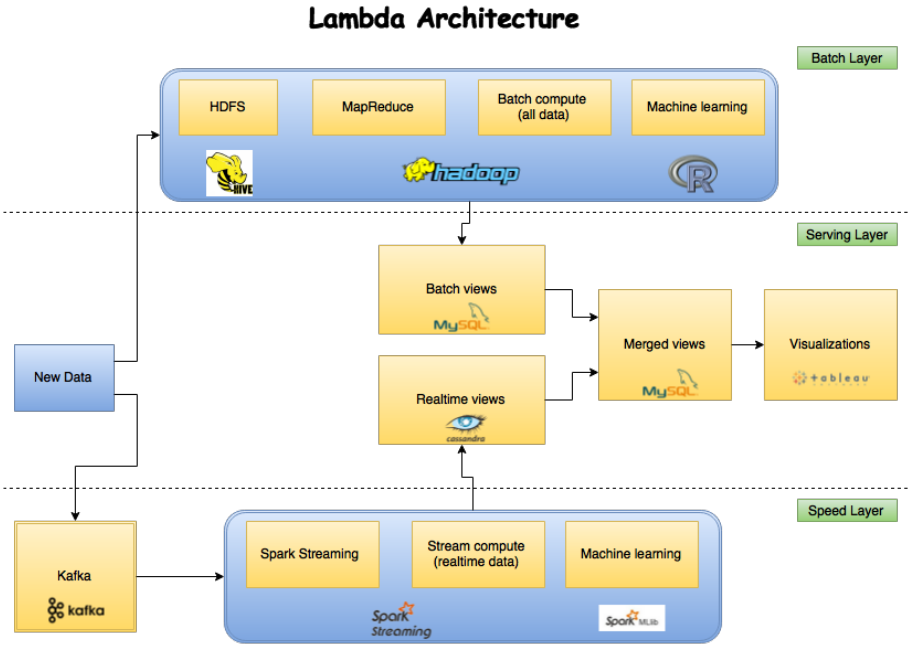

Figure 4.1: Lambda Architecture Model

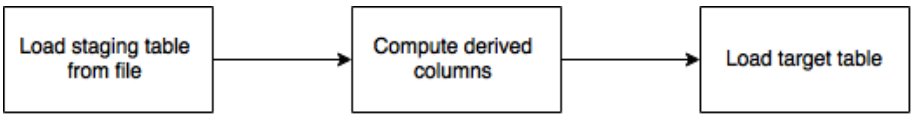

Figure 4.2: Hive data load process flow

4.1.2 Descriptive Statistics

Moving average (Section 3.3.1) and moving standard deviation (Section 3.3.2) are the two descriptive statistical values computed. To efficiently compute this, Hive User Defined Functions (UDFs) are utilized. A Hive UDF helps to extend the functionality written in a high-level programming language (such as Java) and utilize them as functions in Hive Query Language. In this way, UDFs help to encapsulate complex programming logic which cannot be easily expressed using SQL, and at the same time utilizes the parallel processing capabilities of MapReduce.

In this Lambda Architecture batch layer setup, integration between the data storage and processing technologies is designed in such a way as to utilize the best aspects of each. Therefore, the computation of the descriptive

statistics is performed in Hive and MapReduce, and the plotting of the data is performed in R. By integrating Hive and R, the following benefits are achieved.

- Since R is in-memory, it is not well suited for processing *big datasets*. So, the data intensive operations can be moved to MapReduce.
- The power of R libraries can be used to full extent.
- Dashboard applications can be easily developed in R Shiny.

The operational flow of the moving average and moving standard deviation UDFs are shown in Figure 4.3.

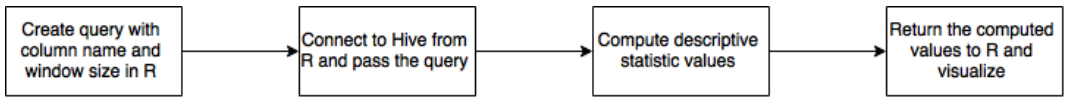

Figure 4.3: Descriptive statistics process flow

4.1.3 Speed Forecasting

Forecasting is one of the most predominant use cases in time series analysis. In Section 3.4, some of the forecasting methods were discussed. The statistical hypothesis testing methods in relation to time series modeling were discussed in Section 3.5. In this section, the implementation of the forecasting models and the testing methodologies are discussed.

A forecasting model to predict the speed of vehicles was developed using ARIMA. Before the actual modeling, the stability of the model is ensured by conducting stationarity test on the time series. Once the model is developed and speed values are forecast, residual analysis is performed to ensure that the model is appropriate.

Figure 4.4 shows the overall operational flow for forecasting and statistical hypothesis testing.

4.1.3.1 Testing for Stationarity

Tests for stationarity were discussed in Section 3.5.2. Augmented Dickey-Fuller test was conducted on the data using the R function *adf.test* [47].

If the time series is not stationary, then it has to be differenced till stationarity is achieved.

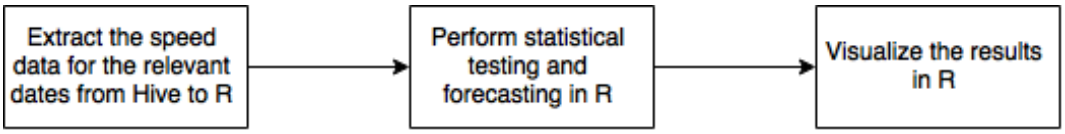

Figure 4.4: Forecasting process flow

4.1.3.2 ARIMA Forecast

An ARIMA forecast model was developed for the time series data using R package *forecast* [30].

4.1.3.3 Residual Analysis

The theory behind residual analysis was covered in Section 3.5.3. Ljung-Box Test (Section 3.5.3.2) was performed on the residuals from the ARIMA forecast model using the R function *Box.test* [41].

The results for forecasting are presented in Section 5.2.2.

4.1.4 Clustering of Vehicles

Clustering of the vehicles is performed based on their length and height. k-means and *EM* are the two clustering methods used. The objective for clustering in the batch layer are the following.

- To show that it is possible to integrate the clustering methods into the batch layer.
- To identify whether the clusters formed relate to other attributes, such as the lane used by the vehicles. Such understanding of the traffic flow will help in developing traffic management best practices.

Before the clustering is performed, the number of clusters, k, needs to be set (see Section 3.6.1). Sum squared of errors method is used for this.

k-means and *EM* clustering methods were utilized. The k-means function from *stats* R package [41] and functions from *EMCluster* package [18] were used.

Figure 4.5 provides an overall process flow of clustering. The results from clustering and the analysis of the clusters are presented in Section 5.2.3.

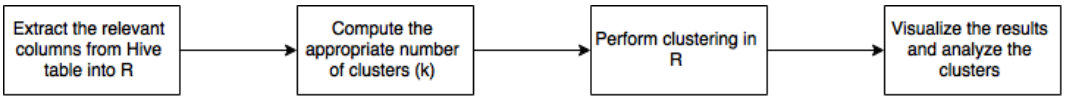

Figure 4.5: Clustering process flow

4.2 Speed Layer

The fundamental idea of speed layer is to perform realtime analysis of the data. A message broker and a streaming data processing framework are needed for this. In this thesis, Apache Kafka (see Section 2.4.2) is used as the message broker and Apache Spark Streaming (see Section 2.4.1) is used as the processing platform. A streaming machine learning use case is also implemented.

4.2.1 Implementation

A Kafka producer is created with a specific topic name to send the data. Spark streaming scripts to perform analysis is executed so as to read the data from the Kafka producer. Machine learning, namely clustering using streaming k-means, is performed in Spark. The results from the computations in Spark are stored in Apache Cassandra for reporting. Figure 4.6 shows the process flow.

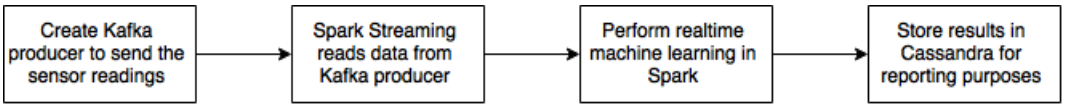

Figure 4.6: Speed layer process flow

4.2.1.1 Streaming k-means

Streaming k-means algorithm was explained in Section 3.6.2. In this section, the implementation details are explored. The results are presented in Section 5.3.1.

In streaming k-means, the model is trained on a selected set of data. This model is further used to cluster the records as they arrive. The vehicles are clustered based on the two attributes of height and length. The process flow is presented in Figure 4.7.

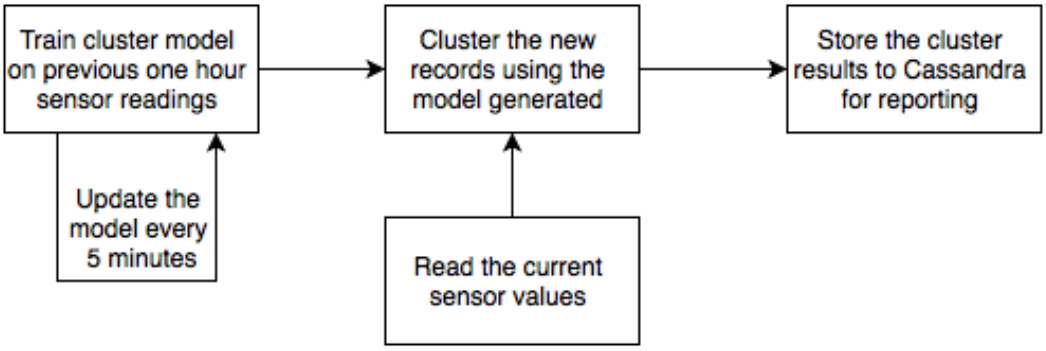

Figure 4.7: Realtime clustering process flow

4.2.1.2 Realtime Views

Apache Cassandra is used to provide realtime views of the cluster results. Table 4.1 lists the values that are stored in Cassandra for further reporting.

Table 4.1: Columns in Cassandra realtime view

Column name	*Data type*
vehicle time	timestamp
sensor id	int
speed	int
length	float
height	float
cluster	int

Queries can be written using Cassandra Query Language to retrieve data from this view and visualize as needed.

4.3 Serving Layer

The serving layer provides an interface to index and query the results from batch layer computations. In this thesis, due to the simplicity of the reporting requirements, we use a MySQL database. The theoretical background is covered in Section 2.3. In this section, the MySQL data model is presented.

4.3.1 Implementation

Section 4.1.4 discusses the cluster analysis performed on the data. Table 4.2 shows the data model created in MySQL to store the cluster results for further analysis and visualizations.

Table 4.2: MySQL data model

Column name	*Data type*
vehicle time	timestamp
Sensor id	int
Height	decimal
Length	decimal
Speed	int
Snow Depth	decimal
Cluster ID	int

Tableau[1] is used for visualizing the data from MySQL tables.

[1]http://www.tableau.com

Chapter 5

Experiments and Results

In this chapter, the experiments from each layer of the Lambda Architecture is showcased and the results are discussed.

5.1 Experimental Setup

The various Big Data frameworks used in this thesis were discussed in Chapter 2. The machine learning tasks and the technological platforms used were discussed in Chapter 3. Before the details of the experiments conducted are presented, the server configuration and the details of the core software components are provided.

Table 5.1 provides the configuration details of the server on which the Lambda Architecture layers are setup and the various analysis tasks are run. Table 5.2 provides the details of the software components that form the core of the Lambda Architecture system developed in this thesis.

Table 5.1: Server details

OS	Ubuntu Linux 12.04 LTS 64-bit
CPU Cores	8
Processors	2 x Intel(R) Xeon(R) E5640 @ 2.67GHz
Memory	8 x 8 GB DIMM DDR3

5.2 Batch Layer

The details of the batch layer and its implementation were covered in Sections 2.2 and 4.1. This section presents the results of the various use cases

analyzed in batch layer.

Table 5.2: Software components

Component Name	*Version*
Apache Hadoop	2.6.0
Apache Hive	0.13.1
Apache Spark	1.4.0
Apache Kafka	2.10
Apache Cassandra	2.10
MySQL	5.5.41
R	3.1.3 (Smooth Sidewalk)
Scala	2.10.3
Python	2.7.3
Tableau	9

5.2.1 Moving Average and Moving Standard Deviation

The moving average and moving standard deviation values were computed using Hive and R. Figures 5.1 and 5.2 visualizes the results for two sensors for a particular day.

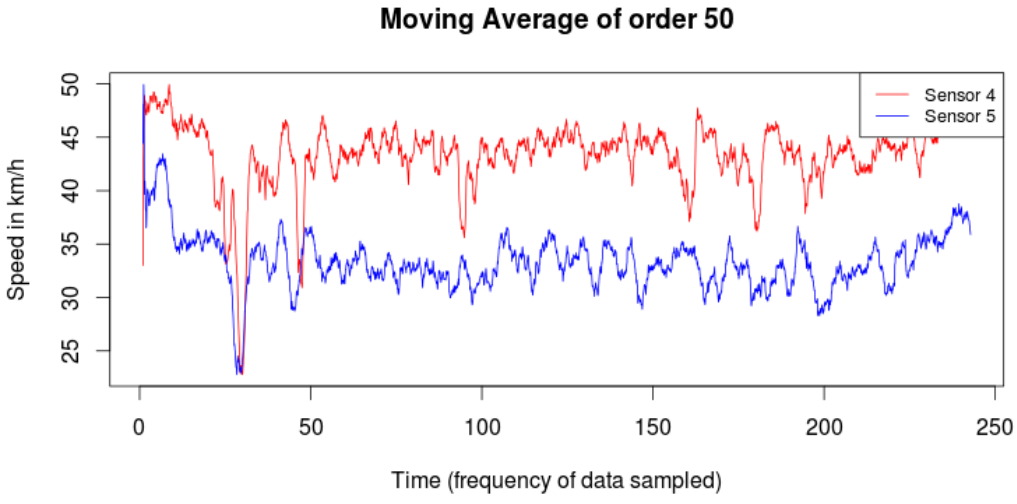

Figure 5.1: Moving Average for Sensors 4 and 5 for 20 March 2015

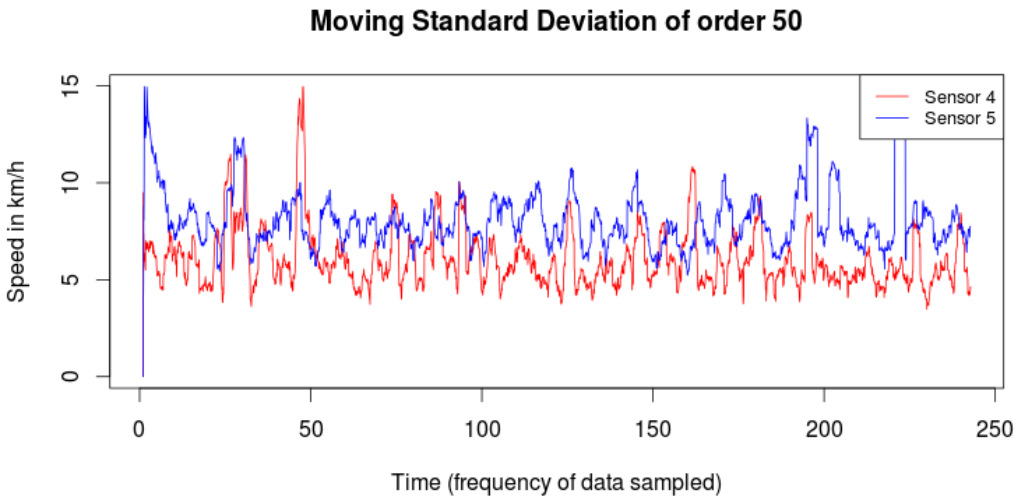

Figure 5.2: Moving Std. Deviation for Sensors 4 and 5 for 20 March 2015

5.2.2 Forecasting Speed

The background for forecasting in time series was discussed in Sections 3.4 and 3.5. In this section, the experiments conducted and the results are presented.

5.2.2.1 Testing for Stationarity

As mentioned in Section 4.1.3, stationarity test was performed on the time series before the forecast models were developed. Augmented Dickey-Fuller test was utilized. The *null* and *alternative* hypothesis are as follows:
Null Hypothesis, H_0 : Series contains a unit root
Alternative Hypothesis, H_1 : Series is stationary

The results are shown in Listing 5.1.

Listing 5.1: Augmented Dickey-Fuller test results

```
        Augmented Dickey-Fuller Test

data:  data.ts
Dickey-Fuller = -5.8624, Lag order = 17, p-value = 0.01
alternative hypothesis: stationary
```

The small p-value suggests that the null hypothesis can be rejected. Therefore, the time series data is stationary and ARIMA modeling can be

performed on it.

5.2.2.2 ARIMA Forecast

Forecasting model was developed using ARIMA. The forecast model details are provided in Listing 5.2.

Listing 5.2: ARIMA model for speed forecast

```
Series: data.ts
ARIMA(1,1,2)(1,0,0)[24]

Coefficients:      ar1      ma1      ma2      sar1
                 0.6164  -1.1749   0.2220  -0.0103
s.e.             0.0341   0.0392   0.0338   0.0140
sigma^2 estimated as 30.73:  log likelihood=-16412.75
AIC=32835.49   AICc=32835.5   BIC=32868.31
```

Figure 5.3 shows the actual observed speed values for a day and the speed values predicted by the ARIMA model. It can be seen that the ARIMA forecast model is able to predict the trends in the data, but the variance of the forecast values is not as good as the observed values.

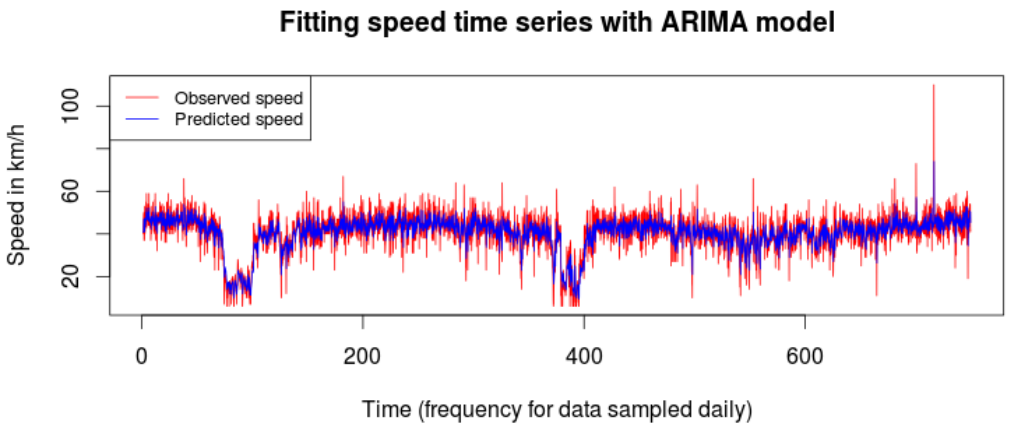

Figure 5.3: Observed speed values and ARIMA model values

Speed values were forecast using this model. Listing 5.3 shows the next 10 future values forecast using the model.

Listing 5.3: Speed forecast values using ARIMA

Point	Forecast	Lo 80	Hi 80	Lo 95	Hi 95
219.4167	44.57822	37.47440	51.68203	33.71387	55.44256
219.4583	45.09718	37.33182	52.86254	33.22108	56.97327
219.5000	45.71006	37.62032	53.79981	33.33787	58.08226
219.5417	45.89180	37.61863	54.16498	33.23908	58.54453
219.5833	46.25938	37.86818	54.65057	33.42615	59.09261
219.6250	46.34061	37.86419	54.81703	33.37704	59.30418
219.6667	46.65724	38.11311	55.20138	33.59012	59.72437
219.7083	46.47300	37.87098	55.07502	33.31735	59.62866
219.7500	46.39616	37.74199	55.05032	33.16075	59.63156
219.7917	46.54641	37.84356	55.24926	33.23655	59.85627

Figure 5.4 shows forecast values along with the historical values for a longer range. It can be noticed that the ARIMA model tends to forecast values which are closer to the mean speed values when the range of forecast is increased.

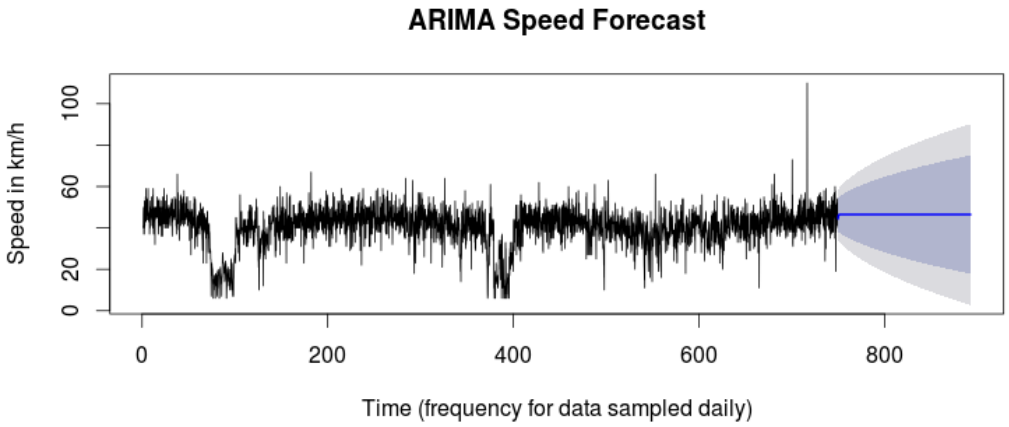

Figure 5.4: Speed forecast values using ARIMA

5.2.2.3 Residual Analysis

Ljung-Box test was used to check for autocorrelation in the residuals. The hypothesis are as follows:
Null Hypothesis, H_0 : The model does not exhibit lack of fit
Alternative Hypothesis, H_1 : The model exhibits lack of fit

The test results are shown in Listing 5.4.

Listing 5.4: Ljung-Box test results

```
Box–Ljung test

data:  speed.forecast.arima$residuals
X-squared = 0.016, df = 1, p-value = 0.8994
```

Since the p-value is high, the null hypothesis cannot be rejected. Therefore, it can be concluded that the ARIMA model developed for speed forecast shows very little evidence of non-zero autocorrelations in the residuals and the model does not exhibit lack of fit.

5.2.2.4 Summary

From the forecast model developed and the statistical testing presented in the previous sections, it can be concluded that by using the speed data of vehicles, we have been able to develop a *fit* forecast model using ARIMA. But the forecast values tend to be closer to the mean values. As shown in Figure 5.3, we have been able to successfully model the pattern of the speed for an entire day. Therefore, we believe that the ARIMA model developed could be used to forecast the speed for the next few vehicles, but not for a longer range.

5.2.3 Clustering Vehicles

In this section, the results for the clustering tasks presented in Sections 3.6 and 4.1.4 are provided.

The first step in clustering is to identify the number of clusters which will explain the data best. The sum of squared errors *elbow* method is utilized. Figure 5.5 plots the sum of squared errors against the number of clusters. It can be seen that the reduction in the error slows down after four clusters. Therefore, we choose the number of clusters (k) to be 4.

k-means and *EM* clustering were performed on the length and height of vehicles. Figure 5.6 shows the clustering output from k-means.

In Finland, the maximum allowed height for a vehicle is 4.4 metres and length is 25.25 metres[1]. It can be seen from Figure 5.6 that there are anomalies in the data. We believe that these outliers are mostly due to malfunctioning of the sensor or due to very little gap between two vehicles which has led the sensor to capture the data for more than one vehicle as the data for one. These outliers are represented by the symbol ✚.

[1]https://www.ely-keskus.fi/en/web/ely-en/

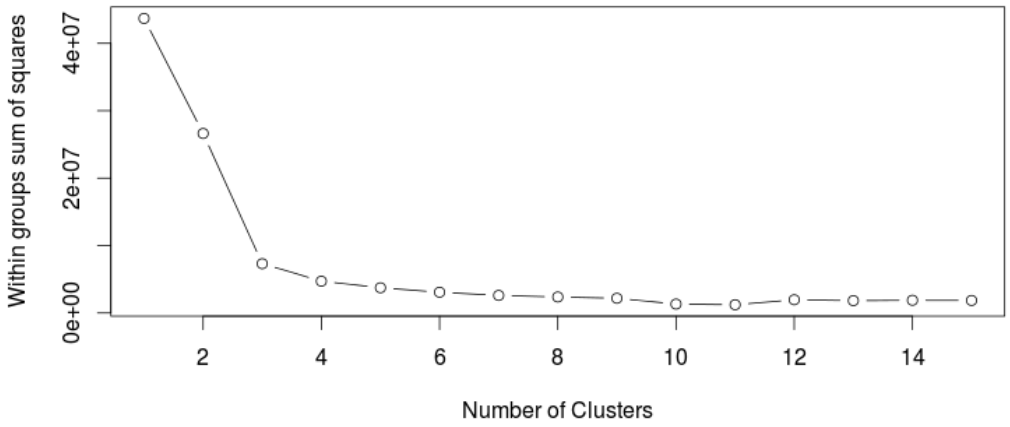

Figure 5.5: Identifying the number of clusters

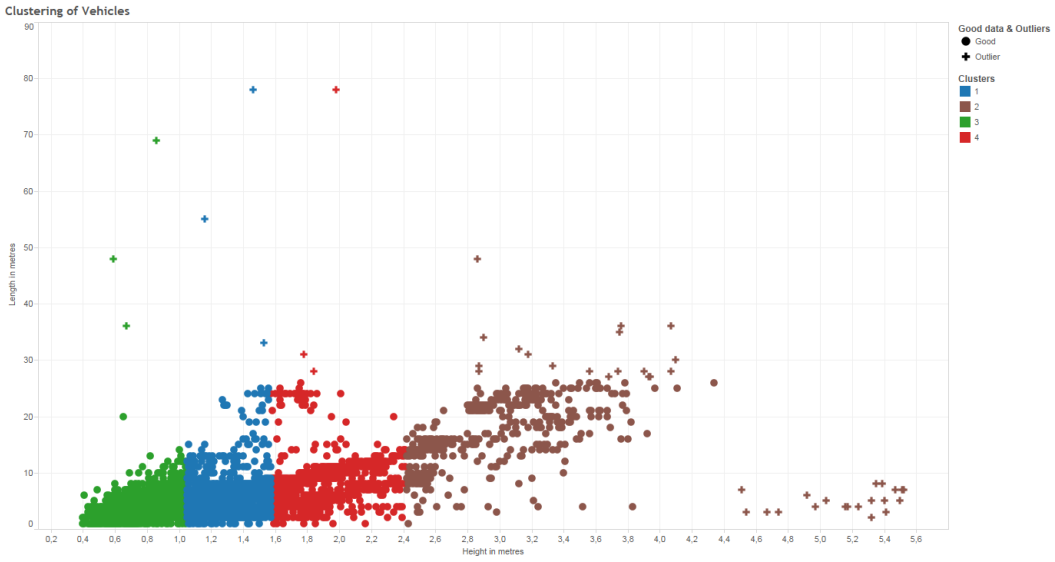

Figure 5.6: k-means cluster results

The cluster results were analyzed to identify relationships between the clusters formed and other attributes in the data. Figure 5.7 shows the distribution of lanes for each cluster. Section 2.6 provided the details of the sensors and the lanes. Lane 4, which is the right lane, tends to be favored by

smaller vehicles whereas the middle lane, Lane 5, is used by slightly bigger and much bigger vehicles. One of the possible reasons is that bigger vehicles do not need to take the exit and continue further, thereby using more of the middle lane and less of the left and right lanes.

The cluster analysis performed along with other traffic attributes can potentially help in traffic flow management, especially in the case of diversions during accidents or road maintenance.

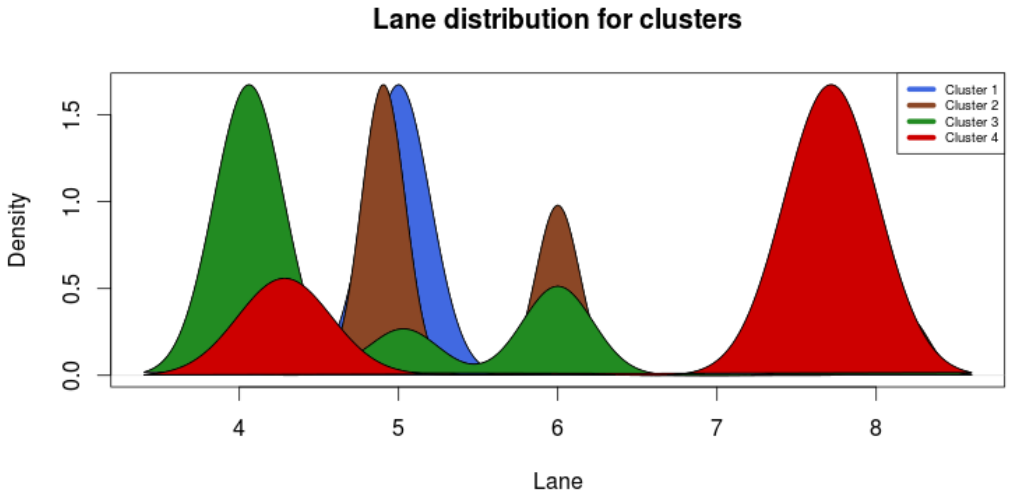

Figure 5.7: Sensor/lane distribution for clusters

5.3 Speed Layer

In Sections 2.4 and 4.2, the theory and the implementation details of the speed layer were presented respectively. In this section, the results of the machine learning use case of realtime clustering of vehicles is provided.

5.3.1 Realtime Clustering of Vehicles

As soon as a sensor reading is received, the vehicle is clustered based on the height and length using the cluster model generated from the past one hour data. The number of clusters is set to 4 (see Section 5.2.3 for details). The realtime cluster results are shown in Figure 5.8.

5.4 Serving Layer

The theory and implementation details of the serving layer were covered in Sections 2.3 and 4.3. In this thesis, merged views are created in MySQL to report the cluster results of the batch data along with the cluster results of realtime data. Figure 5.8 shows a visualization of the data using Tableau.

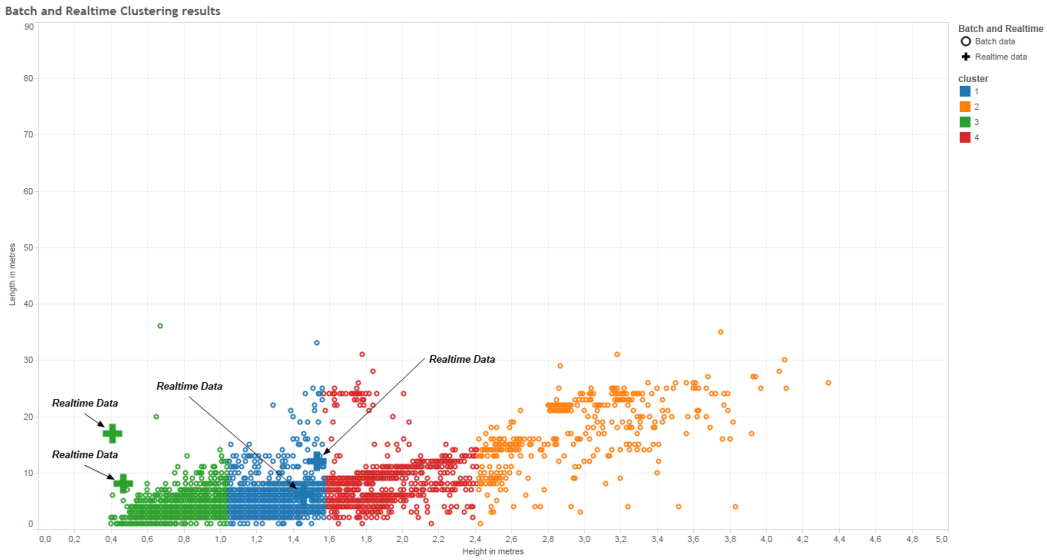

Figure 5.8: Visualizing in serving layer

Chapter 6

Conclusions

As mentioned in the introductory chapter, the major driving forces behind this thesis were the following.

- Explosion of data, especially from sensor devices.
- Development of large scale data storage and processing frameworks.
- Utilization of advanced machine learning techniques in Big Data analytics.

Currently, there is a rapid development and paradigm shift happening in the fields of data processing and reporting technologies. Traditional ETL (Extract, Transform, Load) and business intelligence tools and methodologies tend to be insufficient or non-optimal. This has led to the development of the Lambda Architecture, which has been described in detail in Section 2.1 and Chapter 4.

With the availability of more data storage and processing capabilities, the field of analytics has grown from the traditional approaches of business intelligence to a higher level of analytics, which involves the application of advanced machine learning algorithms and methods. The development of a plethora of open-source platforms for storage and analysis that usually runs on commodity servers, has lowered the cost of adoption of these technologies and in turn has led to a huge interest in this area.

Since the software community has been developing a huge number of solutions, it becomes important to make the right decision and choose the best one which caters to the business needs. This rapidly increasing variety of solutions calls for a central organization to keep track of the various projects, merge similar ones and provide documentation and support. The Apache Software Foundation[1] has been the front-runner in this aspect. But the

[1] http://www.apache.org

current environment is still a bit disorganized and cluttered which makes the decision making process of choosing the best solutions a little complicated.

While developing the Lambda Architecture model in this thesis, this aspect was one of the driving forces for choosing the appropriate solution for each layer. Apart from the functionality offered by a solution, the applicability of implementing it in a production environment was thoroughly considered while choosing the solution. The software components used in this thesis are proven and stable releases, which will be beneficial for anyone trying to productionize this architecture.

Table 6.1 lists some of the major users of the various software solutions discussed in this thesis[2]. It can be noted from this list that almost all of these solutions are used by companies from various industries, and some of them are processing petabytes of data per day. This could be seen as a benchmark of how well these systems will scale and be productionized successfully.

Table 6.1: Big Data solutions and users

Solution	*Prominent users*
Apache Hadoop	eBay, Facebook, LinkedIn, Yahoo!, Quantcast
Apache Hive	eBay, Facebook, Twitter, Scribd
Apache Spark	Alibaba Taobao, Baidu, eBay, Groupon, Yandex
Apache Kafka	LinkedIn, Spotify, Netflix, Cloudflare, Airbnb
Apache HBase	eBay, Adobe, Facebook, Twitter, Yahoo!
Apache Cassandra	Netflix, Comcast, GitHub, GoDaddy
MLlib	Act Now, AsiaInfo, Concur, OpenTable
R	Facebook, Bank of America, Ford, FDA
MySQL	Uber, Walmart, Nokia, Youtube, Bank of Finland
Tableau	Audi, The World Bank, US Army, Target

As mentioned earlier, there are many solutions available for storage and analytics in the Big Data realm. It is imperative that the decision to use a particular solution has to be taken based on the needs of the project. A listing of open-source machine learning software can be found at MLOSS[3], [40] and [44]. For Big Data benchmarks, we refer to AMPLAB[4] and Daytona sort benchmark[5]. For NoSQL databases, we refer to [48] and [1].

Another aspect in this field is the increasing adoption of cloud computing platforms. Amazon Web Services and Microsoft Azure enables anyone

[2]http://pastebin.com/tJ5xG5Aj
[3]http://mloss.org/software/
[4]https://amplab.cs.berkeley.edu/benchmark/
[5]http://sortbenchmark.org

to setup Big Data processing platforms with little effort and low cost. The machine learning offerings such as Microsoft Azure Machine Learning[6] and Amazon Machine Learning[7] brings the power of advanced analytics to everyone.

This thesis has explored the design of the three layers of the Lambda Architecture - *batch layer, serving layer and speed layer*. Open source platforms and solutions which fit the needs of each of these layers were analyzed. Various machine learning applications and statistical hypothesis testing methods were described in detail. The implementation of a Lambda Architecture system and the analytics use cases were explored and results were presented. It can be summarized that a Lambda Architecture system was desgined and implemented and machine learning applications were developed and executed on it.

As more and more home and business users embrace Internet of Things, the storage and analytical options need to be re-evaluated. We believe that this thesis will serve as a good baseline and starting point for further research and development in this area.

[6]http://azure.microsoft.com/en-us/services/machine-learning/
[7]https://aws.amazon.com/machine-learning/

Bibliography

[1] Abramova, V., Bernardino, J., and Furtado, P. Which NoSQL Database? A Performance Overview. *Open Journal of Databases (OJDB) 1*, 2 (2014).

[2] Abramowitz, M., and Stegun, I. A. *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. No. 55. Courier Corporation, 1964.

[3] Alpaydin, E. *Introduction to Machine Learning*. MIT Press, 2014.

[4] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., and Zaharia, M. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), pp. 1383–1394.

[5] Berg, K. L., Seymour, T., and Goel, R. History Of Databases. *International Journal of Management & Information Systems (IJMIS) 17*, 1 (2012), 29–36.

[6] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] Braverman, V., Meyerson, A., Ostrovsky, R., Roytman, A., Shindler, M., and Tagiku, B. Streaming k-means on Well-Clusterable Data. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011* (2011), D. Randall, Ed., SIAM, pp. 26–40.

[8] Brewer, E. A. A certain freedom: thoughts on the CAP theorem. In *Proceedings of the 29th Annual ACM Symposium on Principles of*

Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010 (2010), A. W. Richa and R. Guerraoui, Eds., ACM, p. 335.

[9] Brigadir, I., Greene, D., Cunningham, P., and Sheridan, G. Real Time Event Monitoring with Trident. In *RealStream: Real-World Challenges for Data Stream Mining workshop at European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2013), Prague, September 23th to 27th, 2013* (2013).

[10] Brockwell, P. J., and Davis, R. A. *Introduction to Time Series and Forecasting*. Springer Science & Business Media, 2006.

[11] Brooks, C. *Introductory Econometrics for Finance*. Cambridge University Press, 2014.

[12] Cattell, R. Scalable SQL and NoSQL Data Stores. *SIGMOD Record 39*, 4 (2010), 12–27.

[13] Chan, K.-S., and Ripley, B. *TSA: Time Series Analysis*, 2012. R package version 1.01.

[14] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. Bigtable: A Distributed Storage System for Structured Data (Awarded Best Paper!). In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA* (2006), pp. 205–218.

[15] Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. *shiny: Web Application Framework for R*, 2015. R package version 0.11.1.

[16] Chen, M., Mao, S., and Liu, Y. Big Data: A Survey. *MONET 19*, 2 (2014), 171–209.

[17] Chen, M., Mao, S., Zhang, Y., and Leung, V. C. M. *Big Data - Related Technologies, Challenges and Future Prospects*. Springer Briefs in Computer Science. Springer, 2014.

[18] Chen, W.-C., Maitra, R., and Melnykov, V. EMCluster: EM Algorithm for Model-Based Clustering of Finite Mixture Gaussian Distribution, 2012. R Package, URL http://cran.r-project.org/package=EMCluster.

[19] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM 13*, 6 (1970), 377–387.

[20] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004* (2004), E. A. Brewer and P. Chen, Eds., USENIX Association, pp. 137–150.

[21] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (methodological)* (1977), 1–38.

[22] Dickey, D. A., and Fuller, W. A. Distribution of the Estimators for Autoregressive Time Series With a Unit Root. *Journal of the American Statistical Association 74*, 366a (1979), 427–431.

[23] Drineas, P., Frieze, A. M., Kannan, R., Vempala, S., and Vinay, V. Clustering Large Graphs via the Singular Value Decomposition. *Machine Learning 56*, 1-3 (2004), 9–33.

[24] George, L. *HBase: The Definitive Guide*. O'Reilly Media, Inc., 2011.

[25] Gerstman, B. B. StatPrimer–Version 6.4. Tech. rep., Technical report, San Jose State University, 2010.

[26] Ghemawat, S., Gobioff, H., and Leung, S. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003* (2003), M. L. Scott and L. L. Peterson, Eds., ACM, pp. 29–43.

[27] Gilbert, S., and Lynch, N. A. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News 33*, 2 (2002), 51–59.

[28] Gravetter, F., and Wallnau, L. *Statistics for the Behavioral Sciences*, 9 ed. CENGAGE Learning, 2013.

[29] Hand, D. J., and Krzanowski, W. J. Optimising k-means clustering results with standard software packages. *Computational Statistics & Data Analysis 49*, 4 (2005), 969–973.

[30] Hyndman, R. J. *forecast: Forecasting functions for time series and linear models*, 2015. R package version 6.1.

[31] Hyndman, R. J., and Athanasopoulos, G. *Forecasting: Principles and Practice*. OTexts, 2014.

[32] Jarque, C. M., and Bera, A. K. Efficient tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals. *Economics letters 6*, 3 (1980), 255–259.

[33] Jia, Y., and Shao, Z. A Benchmark for Hive, PIG and Hadoop, 2009. URL https://issues.apache.org/jira/browse/HIVE-396.

[34] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 7 (2002), 881–892.

[35] Lakshman, A., and Malik, P. Cassandra: A Decentralized Structured Storage System. *Operating Systems Review 44*, 2 (2010), 35–40.

[36] Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., and Hornik, K. *cluster: Cluster Analysis Basics and Extensions*, 2015.

[37] Marz, N., and Warren, J. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2015.

[38] Meng, X., Bradley, J. K., Yavuz, B., Sparks, E. R., Venkataraman, S., Liu, D., Freeman, J., Tsai, D. B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., and Talwalkar, A. MLlib: Machine Learning in Apache Spark. *CoRR abs/1505.06807* (2015).

[39] NexR. *RHive: R and Hive*, 2014. R package version 2.0-0.2.

[40] Pop, D. Machine Learning and Cloud Computing: Survey of Distributed and SaaS Solutions. *Institute e-Austria Timisoara, Tech. Rep 1* (2012).

[41] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[42] Rajaraman, A., Ullman, J. D., Ullman, J. D., and Ullman, J. D. *Mining of Massive Datasets*, vol. 77. Cambridge University Press, 2012.

[43] Sallam, R. L., Hostmann, B., Schlegel, K., Tapadinhas, J., Parenteau, J., and Oestreich, T. W. Magic Quadrant for Business Intelligence and Analytics Platforms. *Gartner* (2015).

[44] Singh, D., and Reddy, C. K. A Survey on Platforms for Big Data Analytics. *Journal of Big Data 2*, 1 (2015), 1–20.

[45] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. Hive - A Warehousing Solution Over a Map-Reduce Framework. *PVLDB 2*, 2 (2009), 1626–1629.

[46] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., and Ryaboy, D. V. Storm@twitter. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014* (2014), pp. 147–156.

[47] Trapletti, A., and Hornik, K. *tseries: Time Series Analysis and Computational Finance*, 2015. R package version 0.10-34.

[48] Tudorica, B. G., and Bucur, C. A comparison between several NoSQL Databases with comments and notes. In *Roedunet International Conference (RoEduNet), 2011 10th* (2011), IEEE, pp. 1–5.

[49] Venables, W. N., Smith, D. M., and Team, R. C. An Introduction to R, 2015.

[50] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012* (2012), pp. 15–28.

[51] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010* (2010), E. M. Nahum and D. Xu, Eds., USENIX Association.

[52] Zivot, E., and Wang, J. *Modeling Financial Time Series with S-Plus®*, vol. 191. Springer Science & Business Media, 2007.